CS10720 Problems and Solutions

Thomas Jansen

Today: Page Rank
Data Compression

April 21st

Plans for Today

PageRank

Algorithm to compute PageRank values (with error $< \varepsilon$) by approximating stationary probabilities for 'random surfer'

Notation

- set of all web pages: V
- number of all web pages: $n = |V|$
- number of different links from v somewhere: $L(v)$
- set of pages with links to v: $I(v)$
- probability of 'restart': r ($1 - r$ called damping factor)
- current estimate of the PageRank of web page $v \in V$: $\mathsf{PR}(v)$

1. For all $v \in V$ set $\mathsf{PR}(v) := 1/n$.
2. Do
3. Set $\Delta := 0$.
4. For each $v \in V$ do
5. $\mathsf{PR}_{\mathsf{new}}(v) := \frac{r}{n} + (1 - r) \cdot \sum_{w \in I(v)} \frac{\mathsf{PR}(w)}{L(w)}$
6. if $|\mathsf{PR}_{\mathsf{new}}(v) - \mathsf{PR}(v)| > \Delta$ then $\Delta := |\mathsf{PR}_{\mathsf{new}}(v) - \mathsf{PR}(v)|$
7. For each $v \in V$ do
8. $\mathsf{PR}(v) := \mathsf{PR}_{\mathsf{new}}(v)$
9. Until $\Delta < \epsilon$

PageRank (in English)

1. Set PageRank value for all pages to $1/$(number of pages) initially.
2. Work in rounds in the following way:

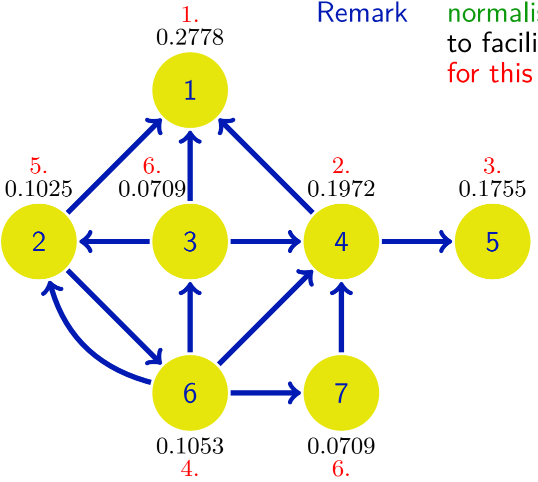
4.-5. Compute the new PageRank value for v as $r/$(number of pages) plus, for each page with a link to v, $(1-r)$ times that page's PageRank value divided by the number of different links leaving it.

6. Keep track of the greatest change in PageRank values.

9. Stop when this difference decreases below ε.

1. For all $v \in V$ set $\mathsf{PR}(v) := 1/n$.
2. Do
3.   Set $\Delta := 0$.
4.   For each $v \in V$ do
5. $\mathsf{PR}_{\mathsf{new}}(v) := \frac{r}{n} + (1-r) \cdot \sum_{w \in L(v)} \frac{\mathsf{PR}(w)}{L(w)}$
6.   if $|\mathsf{PR}_{\mathsf{new}}(v) - \mathsf{PR}(v)| > \Delta$ then $\Delta := |\mathsf{PR}_{\mathsf{new}}(v) - \mathsf{PR}(v)|$
7.   For each $v \in V$ do
8. $\mathsf{PR}(v) := \mathsf{PR}_{\mathsf{new}}(v)$
9. Until $\Delta < \epsilon$

PageRank Tiny Example



Remark normalise so that sum equals 1 to facilitate comparison for this slide only

Intuition for Convergence

Consider 'web graph' without sinks
(i. e., without pages without outgoing links)
because for those graphs no normalisation necessary

Observations

- initially, PageRank value equals probability for uniform distribution
- after one round, PageRank value equals probability for model
  1. with probability r, select new random page
  2. with probability $1 - r$, follow random link
- in each round, probability mass is redistributed in the same way
- iterating this over many rounds leads to stable distribution

About PageRank

Problem PageRank is relatively expensive to compute

Solution re-calculate ranks only occasionally

Problem PageRank value is sensitive to manipulations
when someone sets up large number of web pages with links
to push up PageRank value of some target page

Solution explicitly punish such ‘link farms’
(done by Google at least since 2011)
and/or use other metrics to determine ranking
(done by Google in several secret updates)

Fact PageRank is very useful way beyond ranking web pages
e. g., ranking ‘who to follow’ on Twitter, noise reduction
(e. g., in bioinformatics), support debugging of complex systems,
traffic prediction, book ranking (for tagged books), . . .

Fact PageRank still subject of research (e. g., A. D. Sarma, A. R. Molla, G. Pandurangan, E. Upfal (2015): Fast distributed PageRank computation. *Theoretical Computer Science* 561:113–121)

A Tiny (and completely unrealistic) Example

Consider some introductory animation
(two minutes long, even without any sound, but in colour)

Mostly Realistic Assumptions

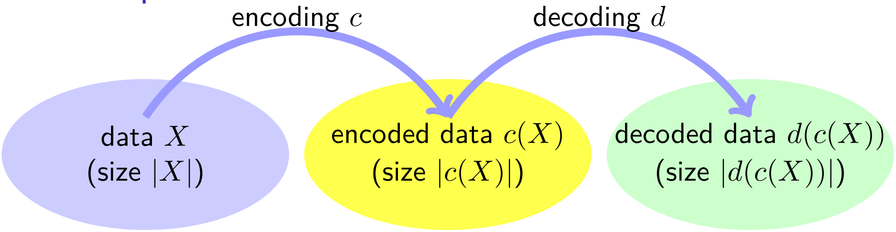
- screen resolution 1280×800
- 3 bytes per pixel to encode colour
- 25 frames per second

Simple Calculation
$1280 \times 800 \times 3$bytes $= 3000$KB per frame
3000KB $\times 25 \approx 73.24$MB per second
73.24MB $\times 2 \times 60 \approx 8789$MB
≈ 8.58GB for the animation

Observation a two-minute long fullscreen animation without sound
does not fit on a DVD

Consequence compression needed

Data Compression



Safe Assumptions
data is ‘text’ over fixed finite alphabet Σ (sometimes $\Sigma = \{0, 1\}$)
encoding $c(X)$ is ‘text’ over alphabet $\{0, 1\}$

	lossless compression	lossy compression
property	$\forall X : d(c(X)) = X$	usually $d(c(X)) \neq X$ but $d(c(X)) \approx X$, of course
desirable	$\vert c(x)\vert$ small	$\vert c(x)\vert$ small even smaller, of course
examples	bzip2, compress, zip, gif	jpeg, mpeg, mp3

Limitations of Lossless Compression

Remember desire compressed size $|c(X)|$ small
(measured in compression rate $|c(X)| / |X|$
or compression factor $|X| / |c(X)|$)

How small can that be?

Observation general lower limit cannot exist
Example fixed text X' over alphabet $\Sigma = \{0, 1\}$

encoding $c(X) = \begin{cases} 0 & \text{if } X = X' \\ 1X & \text{otherwise} \end{cases}$

decoding $d(bX) = \begin{cases} X' & \text{if } b = 0 \\ X & \text{otherwise} \end{cases}$

is lossless compression
with optimal compression rate $1/|X'|$ for X'
and works for arbitrary X'

Observation silly example

Elementary Insights

Insight 1 any fixed text can be losslessly encoded using only 1 bit (by means of the silly example)

Insight 2 for any lossless compression scheme c/d
there exists a text $X \in \{0,1\}^*$ with compression rate ≥ 1
Observation c needs to be injective
(meaning $\forall X_1 \neq X_2 \colon c(X_1) \neq c(X_2)$)
Consequence X_1 with $|c(X_1)| < |X_1|$
$\rightsquigarrow \exists X_2 \colon |c(X_2)| > |X_2|$
(pigeon hole principle)

Consequence for non-trivial bounds some assumptions needed

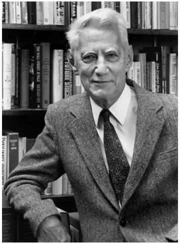
Remark strong assumptions $\rightsquigarrow$ results easy to obtain
weaker assumptions $\rightsquigarrow$ results harder to get

(A tiny bit of) Information Theory

Assumption text $X \in \Sigma^*$ comprises of letters $s \in \Sigma$ with each s occuring with fixed, independent probability $\mathrm{Prob}(s)$

Facts

- such source has entropy $-\sum_{s \in \Sigma} \mathrm{Prob}(s) \log \mathrm{Prob}(s)$
- average coding length bounded below by entropy



1916–2001

Remark complete independence is crude assumption
$\rightsquigarrow$ weak bounds

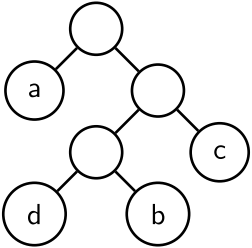
Lossless Encoding Letter-Wise

Remember lossless $\widehat{=} \; \forall X : d(c(X)) = X$

letter-wise $\widehat{=} \; \forall X = x_1 x_2 \cdots x_l : c(X) = c(x_1)c(x_2)\cdots c(x_l)$

Example (we cover) prefix codes, in particular Huffman coding

Huffman Coding example



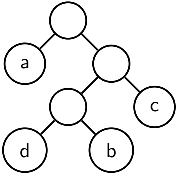
for

s	$c(s)$
a	0
b	101
c	11
d	100

Facts Huffman coding has
optimal expected length and
expected length $\leq$ entropy $+ 1$

Computing Huffman Codes

Remember example Huffmann coding



for

s	$c(s)$
a	0
b	101
c	11
d	100

Algorithm ComputeHuffmanTree

Input Σ and $\mathrm{Prob}(s)$ for all $s \in \Sigma$

Output Huffman code ($\widehat{=}$ tree)

1. For all $s \in \Sigma$ create root node with weight $\mathrm{Prob}(s)$.
2. While number of trees > 1
3. Select T_1, T_2 with minimal weights w_1, w_2.
4. Create new tree with empty root, left sub-tree T_1, right sub-tree T_2, weight $w_1 + w_2$.
5. Remove T_1 and T_2.

Using Huffman Coding

Observations

- given Σ and $\mathrm{Prob}\,(s)$, Huffman tree easy to compute
- given tree, encoding and decoding easy

In practice two options

1. use static Huffman code (requires useful probabilities)
2. compute new Huffman tree for each text
   (requires 'store tree with data', implies overhead)

Example actual application Fax (group 3)

- read page row-wise
- compute run length encoding (RLE)
- use fixed Huffman code for RLE

Summary & Take Home Message

Things to remember

- ranking ideas: popularity (i. e., number of links) and importance (i. e., rank of linking pages)
- idea: random surfer
- PageRank algorithm
- PageRank manipulation
- PageRank applications beyond search
- need for compression
- lossless compression
- Huffman encoding

Take Home Message

- PageRank is a relatively simple and extremely powerful and versatile ranking algorithm for graphs.
- Simple ideas can help earn lots of money but it's hard to recognise a good idea before it happens.
- Compression is a fascinating topic with practical applications